

**T**he problem was to write a program that would answer the question "Can I get from station x to station y without changing?" This only appears to be a difficult question if you read more into it than necessary. The sophisticated approach is to see it as a problem in searching a graph, ie the map of the underground system, to see if there is a direct connection between two nodes, ie the tube stations. This would be the problem if it wasn't for the fact that the graph being searched is fixed, apart from new tube lines being added infrequently, and well known. Because this is the case, you can easily precompute the answers and store them in a 2D lookup table.

Most programmers have come across the idea of a lookup table, but most of us tend to ignore it even when it would make problems simple. In other words, all that is needed to solve this problem is a 2D array, c say, such that `c(station1,station2)` is true if the stations can be travelled between without changing, and false otherwise. The only difficulty with this scheme lies in the indexing of the array on station names, and there are a range of methods for doing this. But this apart, a solution is simply:

**IF c(station1,station2) THEN connected ELSE not-connected**

At this point I can hear readers saying "What about extending the program to answer other questions?" Well, because the graph is fixed, a lookup table can be expanded to answer other questions. For example, you could include the name of the station or stations where a change has to be made, the best route, the cost and so on. The only problem you might have is in finding enough memory, but if you don't store the answers to difficult questions such as the best route then you have the problem of finding enough processor power to answer the question in real-time. It's the traditional trade-off of memory against time.

Some of the solutions submitted this month did use a lookup table, either directly as a 2D array or indirectly in the form of some sort of coding. However, a surprising number of entries attempted to solve the problem via a search or some other complex manipulation of 'station records'.

Phil Perry used a 0, 1 lookup table in *ST Basic* to record which lines each station is on and then went on to point out that a more difficult problem is to give the

# The Tube connection

*Mike James and Kay Ewbank pick the best of your entries from the last Challenge and talk backwards*

changes. Not if you use a lookup table for this too! Bryan Wood used a bit code to indicate which line each station was on in such a way that bitwise ANDing of the two codes gives True if they are each on the same line. Edward Creighton used a well-structured *BetaBasic* to achieve the same result. N Griffith used the same method in *BBC Basic* but also used a hashing technique to find stations in the lookup table. Martin Vlietstra used a quick sort and a binary search for the same job. Vincent Hardman used a binary representation but worked in decimal for initialisation purposes.

Pascal allows you to use the same sort of bitwise coding without being aware of it. Barry Mault made good use of Pascal sets and set intersection to find lines that stations had in common – this is just a more abstract form of the same method, where you read set

intersection for bitwise AND. Philip Hunt and Paul Schofield submitted Smalltalk entries, the first that we have had. The object-orientated method may be taking off after all! They created a serious problem for us because Smalltalk can be highly interactive, rather than procedural, and comparing such programs with conventional programs is difficult.

## Search algorithms

Ian Taig writes that he was surprised that the problem was described as difficult and then presents a program in EGERIA – an expert system shell language – to solve the problem. The program is compact and clear but it makes use of the built-in sophisticated search algorithm of the expert system. This isn't a criticism, just an observation about perceived complexity!

There were also an unusually

high number of Prolog entries, presumably because the programmers spotted the fact that the problem could be cast as a graph search. It is well known that Prolog contains a powerful default search algorithm, so what could be more natural? Among the nicer programs were Neil Harris' well-structured and commented *Turbo Prolog* entry and J Wood's *SD Prolog* program, which also listed possible routes.

James Manzari writes from Switzerland to submit an XLISP solution which processes lists of stations on each line – the Swiss underground system, of course. Also from Switzerland were Paul Schofield's entry in Smalltalk and B Hunt's in Basic, using bit manipulation via decimal arithmetic.

## Exotic language

Just to add to our exotic language collection (EGERIA noted above) David Sweet gave us a program SCL – ICL 2988 mainframe System Control Language. SCL looks vaguely Algol-like and the program did the job well. The UNLESS conditional looked useful! Roger Cope reminded us that MUMPS is still going strong. In case you are worrying, MUMPS is a programming language found on DEC machines and others. It is slightly Basic-like but seems to allow string indexes to arrays, which makes the problem much easier.

G Smith submitted a very complete Forth solution which proves that Forth can be written clearly. A good Cobol solution was submitted by Robin Hillman.

## Database solutions

The number of people who tackled the problem using a database was very high. M Smith used *TAS Plus* to produce a compact program. Ewan Bowman used *Clipper* to the same effect and comments that he tried to extend the program to provide a route – he now appreciates the subtleties of *Autroute*!

Finally, Ted Creighton comments that he believes we are looking for a well-structured and, well, boring program as the winner. There is nothing to say that ingenious schemes cannot be built into clear programs. We judge the competition on a combination of style and method. Ted's program was well-structured, but never boring. Thank you all for your entries. Everyone mentioned here wins a *Shopper* guru mug. Our apologies to all those entrants who didn't get a mention – there were just too many good programs. ■

## THIS MONTH'S CHALLENGE

### The reverse cat problem

This month's Programmer's Challenge is a little harder than it looks, as many entrants to the *Shopper* Show discovered to their cost! This is the last of the three challenges set at the show.

All you have to do is write a program that will accept a sentence from the user and print it out in reverse word order. For example, if you type:

**The cat sat on the mat**

then your program should print:

**Mat the on sat cat the**

Simple? Well, the extra condition that the reversal should be performed 'in place' if this is possible makes it a bit more difficult. That is, try only to use the array or string variable that the input is read into. Also, methods based on screen positioning using cursor control are banned. You have to read in the sentence, then reverse the word order without making a copy, and finally print out the result. The entries will be judged on the method used to achieve the result and the quality and clarity of the coding.

Use any language of your choice and include a very brief note at the start giving a rough idea of the approach you have used. Please send listings only – no disks please. The closing date is 21st May 1990 and all entrants who get an honourable mention will receive a *Computer Shopper* mug. Send your entries to:

**Reverse cat problem, Programmer's Challenge,  
Computer Shopper, 14 Rathbone Place, London W1P 1DE.**